

Behavior recognition and analysis in smart environments for context-aware applications

Radosław Klimek

AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
rklimiek@agh.edu.pl

Abstract. Providing accurate/suitable information on behaviors in smart environments is a challenging and crucial task in pervasive computing where context-awareness and pro-activity are of fundamental importance. Behavioral identifications enable to abstract higher-level concepts that are interesting to applications. This work proposes the unified logical-based framework to recognize and analyze behavioral specifications understood as a formal logic language that avoids ambiguity typical for natural languages. Automatically discovering behaviors from sensory data streams as formal specifications is of fundamental importance to build seamless human-computer interactions. Thus, the knowledge about environment behaviors expressed in terms of temporal logic formulas constitutes a base for the reactive and precise reasoning processes to support trustworthy, unambiguous and pro-active decisions for applications that are smart and context-aware.

Keywords: Unified logical framework; sensorized environment; context-awareness; temporal logic; semantic tableaux.

This is a pre-print author's version of the paper: R. Klimek: Behavior recognition and analysis in smart environments for context-aware applications. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2015), October 9–12, 2015, Hong Kong*, pp. 1949–1955. IEEE Computer Society 2015. Available at: DOI:10.1109/SMC.2015.340 or <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7379472>

1 Introduction

Nowadays smart spaces are filled with different sensors and sensor-like equipments. A *sensor* is a device that detects events or changes from a physical environment, that is a device which is sensitive to a physical stimulus. These sensors might constitute the IoT spaces (*Internet of Things*) in which objects with unique identifiers create their own scenarios and interactions. On the other hand, the decisive feature of smart spaces is *context-awareness* which stands for the capabilities to examine changes in the environment and to react to these changes adequately. Important aspects of context might be: where you are, who

you are with, and what resources are nearby. In other words, context is “...any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves” [7]. In software engineering context-awareness means sensing and reacting on the environment. Sensing and context understanding are necessary and of critical importance to pro-active decisions which should be interpreted into domain-relevant concepts and situations.

Formal logic allows assertions about actions and behaviors using accurate and precise notations, eliminating ambiguity common to other languages. “Logic has simple, unambiguous syntax and semantics. It is thus ideally suited to the task of specifying information systems” [5] showing the form of an argument to be valid or invalid. Knowledge about arguments enable achieving clear thinking and relevant arguments.

The contribution of this paper is a novel and unified logical-based framework to deploy automatic methods for the behavior recognition and its reliable knowledge representation through the formalism of temporal logic. It allows to support reactive analysis of logical satisfiability, in order to obtain trustworthy decisions for the dynamically changing smart environment. Decisions of a system are transparent for users/inhabitants and satisfy the assumption of context-awareness and pro-activity. It is demonstrated that this logical framework is expressive enough. It is also demonstrated that on-line logical reasoning is suitable for sensor data streams. The semantic tableaux method for temporal logic as a reasoning procedure is considered. The architecture of a software system (see Figure 5) is proposed, as well as algorithms (see Algorithm 1 and Algorithm 3) to generate and interpret logical specifications. The simple yet illustrative examples are provided, see Formulas (2) and (3) for Algorithm 1 and the discussed example for Algorithm 3 at the end of Section 5, as well as related motivating examples in Section 3. To the best of our knowledge, this paper presents the first formal study of both the reactive behavior recognition and deductive-oriented analysis for context-aware applications over sensor networks. On the other hand, this research opens some new directions, especially related to implementation and experiments.

There are many works considering behavior analysis in pervasive computing. A survey for human activity is provided in the fundamental work [1]. Features, representations, classification models, and datasets are surveyed. Work is comprehensive and discusses many important aspects of the domain. This paper refers to single-layered approaches as considered in [1]. A survey of activity recognition for wearable sensors is provided in work [15]. A taxonomy according to aspects of response time and learning scheme is introduced. A couple of systems are qualitatively compared due the mentioned aspects, as well as some other ones. Formal logic approaches, except for the fuzzy logic, are not considered. Behavior recognition in smart homes is a topic in work [6], whose approach influenced in some way this paper, however, models base on Hidden Markov Models, which constitutes a different approach in comparison to this

one. In work [3] a hierarchical framework for human activity recognition is presented, however, the framework focuses on video based activity recognition. The method of rather manual transformation into logical rules, is done in an off-line manner, and reasoning based on the resolution is proposed. The aim is to discover a semantic gap between the low level (data) and the high level (human understanding). In work [4] a similar approach is presented but formalization is based on an adaptation of temporal relations from the Allen's temporal interval logic, and the reasoning process is not considered. Apart from the issue of a hierarchical approach, these works influence this paper in such a way that the formalization of the observed (human) activities is made on the basis of formal logic. This paper follows work [12] which concerns on-the-fly modeling logical specifications and observing behaviors of users/inhabitants, in other words, logical specifications are understood as knowledge about user preferences. Work [8] proposes patterns for a property specification and is considered in a more detailed way in the following Sections of this paper, especially when discussing the so called learning-based approach. Work [16] discusses possibilities of using temporal logic and model checking for the recognition of human activities. This paper is relatively close to the work, however, a deduction based approach is proposed. The novel aspects are unified logical framework, basing on a purely logical approach, and deductive-based reasoning processes to obtain pro-active decisions.

2 Preliminaries

A context model that consists of three layers is shown in Figure 1, c.f. also [12]. It contains different sensor devices which are distributed in the whole physical

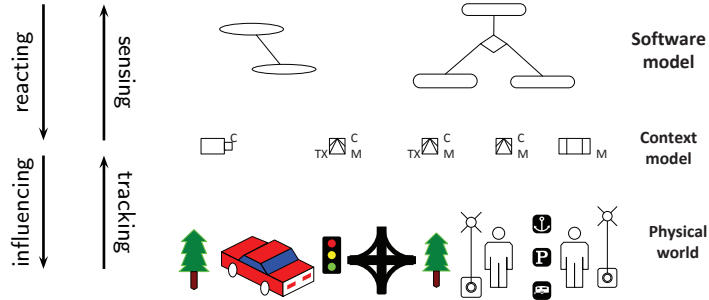


Fig. 1. A three-layer context model for a smart environment

area. It also refers to the concept of *Ambient Intelligence* (AmI), i.e. electronic devices that are sensitive and responsive to the presence of humans/inhabitants. Smart applications must both understand context, that is be context-awareness, and provide pro-activity, that is act in advance to deal with future situations,

especially negative or difficult ones. A context-aware system is able to adapt its operations to the current context without explicit user intervention.

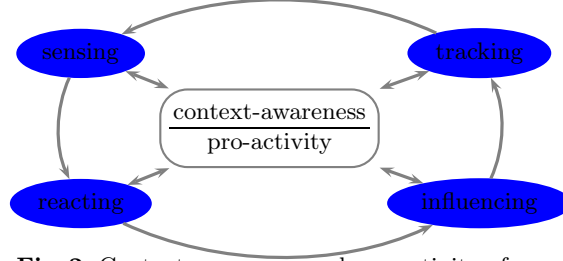


Fig. 2. Context-awareness and pro-activity of apps

The dynamic nature of context models' analysis is shown in Figure 2, i.e. supplementing Figure 1, where different phases are repeated periodically to sense behaviors and to generate proper system's reactions enabling context-awareness and pro-activity of applications which operate in a smart environment.

Temporal Logic, and *Propositional Linear Time Temporal Logic* PLTL considered here, is a branch of formal logic with statements whose valuations depend on time flows [18]. The reasoning method of *semantic tableaux* is well known in classical logic but it can be applied in temporal logic [9,11]. The method provides *truth trees*. The *branch* of a tree is a set of nodes/formulas connecting a node with a descendant. Semantic tableaux is also a *decision procedure* providing, through *open* branches (that is, not containing complementary pair/pairs of atomic formulas, e.g. f and $\neg f$) and *closed* branches (that is, containing complementary pair/pairs of atomic formulas), the binary answer Yes-No as a result of an inquiry.

Corollary 1. *If F is an examined formula and Δ is a truth tree build for a formula, then the semantic tableaux method gives answers to the following questions related to the satisfiability problem:*

- formula F is not satisfied iff the finished $\Delta(F)$ is closed;
- formula F is satisfiable iff the finished $\Delta(F)$ is open;
- formula F is always valid iff finished $\Delta(\neg F)$ is closed.

The proof follows directly from the semantic tableaux method.

3 Motivating examples

Let us consider some examples to illustrate the approach and provide some motivation. A basic distinction two approaches regarding method of building logical specification is introduced:

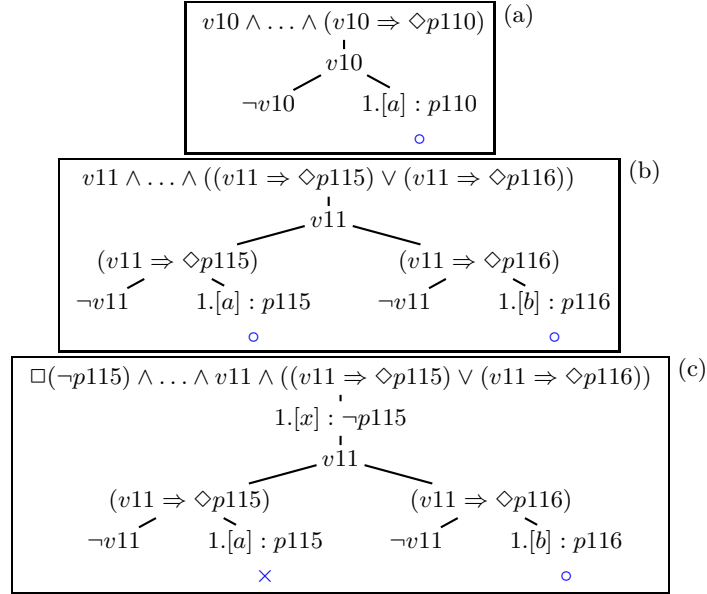
1. *model-based* – the case occurs when logical specifications (models) for context-aware systems are prepared in advance; in other words, the initial specification is not empty, but new events may affect a particular specification leading

to its modification, it can be used in a decision/reasoning process without any change, but logical specification can also be dynamically expanded/rebuilt when the system operates, see the evacuation example below;

2. *learning-based* – the case occurs when logical specifications are build on-line, that is in real-time, during normal operation of a context-aware system; in other words, the initial specification is initially empty, and when new events occur, logical specification is built/rebuilt, and at any time it can be used in decision-making processes see work [12] or the second example below.

The first example discusses an evacuation situation, i.e. people are located inside risk areas (e.g. buildings or sport stadiums) and a dangerous situation occurs. Context-aware and smart systems should help inhabitants/people by providing trustworthy information about evacuation paths. The evacuation plan, expressed as a logical specification Σ , and understood as a set of temporal logic formulas, must be prepared in advance. (This is a reverse situation comparing other hypothetical cases where logical specifications might be built on-line i.e. when the system operates.) Formulas describe possible and recommended actions/transitions during the evacuation process. After the evacuation process has been started and is being carried out, dynamically changing situations, e.g. fire on a passage, may require extension of Σ introducing new formulas describing new situations. It is done by software agents observing changes in particular areas. (Graph-based description might contains nodes with different attributes, such as entrances to corridors or staircases and edges that connect different areas.)

Let $V = \{\dots, v10, v11, \dots, p110, p115, p116, \dots\}$ are places and passages of a building for which the evacuation plan is to be prepared. $\Sigma = \{\dots, v10 \Rightarrow \Diamond p110, \dots\}$ is a (small) fragment of the evacuation plan expressed in terms of temporal logic formulas. When new objects appear in place $v10$ ($v10$ is satisfied), then the reasoning process starts, see Figure 3.a. The open branch (\circ) of the tree provides *literals*, that is atomic formulas or their negations, $v10$ and $p110$ that satisfy the initial formula that consists of satisfied $v10$ and conjunction of all formulas that belongs to Σ . It allows to identify formula $v10 \Rightarrow \Diamond p110$ that describes the next supporting people action for a particular place, as a part of an evacuation process. Another situation is shown in Figure 3.b. Let $\Sigma = \{\dots, ((v11 \Rightarrow \Diamond p115) \vee (v11 \Rightarrow \Diamond p116)), \dots\}$ is another fragment of an evacuation plan showing the choice of escape routes. New objects which appear in place $v11$ involve the reasoning process that provides through two open branches, two subsets of literals $v11$ and $p115$, and also $v11$ and $p116$. It means that two different actions are possible, i.e. $v11 \Rightarrow \Diamond p115$ or $v11 \Rightarrow \Diamond p116$. In the last case, see Figure 3.c, the extension of logical specification Σ is discussed. Supposing that the dynamically changing situation, e.g. fire, forces the closure of passage $p115$. It leads to the need of extending the logical specification by a new formula $\Box(\neg p115)$, i.e. $\Sigma := \Sigma \cup \{\Box(\neg p115)\}$. Thus, every reasoning process for $p115$ leads to the closed branch (\times), i.e. the contradiction. It means that the “fired” passage will never be proposed as an action for the evacuation procedure. This example is also discussed in a more formal way after Algorithm 3 in Section 5.

**Fig. 3.** The product of reasoning – sample truth trees

The above considerations should be supplemented with the following information. The accepted decomposition procedure in Figure 3, as well as labeling, refers to the first-order predicate calculus provided in [9]. In some cases, the outer operator \Box is omitted to simplify considerations/formulas, in other words, for example, one should write down $\Box(v11 \Rightarrow \Diamond p116)$, however, the well-known rules of generalization/particularization justify the simplified notation. Reasoning engines have become more available in recent years, c.f. [17], however, selection of an appropriate existing prover is not in the scope of this paper.

Another example might refer to the situation when logical specification Σ , interpreted as knowledge about user/inhabitant behaviors, is built on-line, i.e. initially $\Sigma = \emptyset$, and then, observing present users' behaviors, new temporal logic formulas for particular objects/users are added to set Σ . Work [8] discusses methods of obtaining logical specifications from a natural language. The method is based on pattern recognition. The consideration in this paper provides a method/idea to obtain logical specifications from a (technical) language of physical sensors/signals which is less complex when comparing it to a natural language. Some sample patterns for a "sensor language" are provided in Figure 4. Registered (atomic) events for every user might comprise a label for a physical node (e.g. the presence in a node) and time for the event occurrence (i.e. the time stamp), e.g. $\langle p210, t2014.08.14.21.56.00 \rangle$. These elementary events are translated into logical specifications when analyzing time of the events and employing (pre-defined) patterns. If logical specification Σ is built, then the pro-active decision might be taken when new event, say gt , occurs and is considered as a kind of

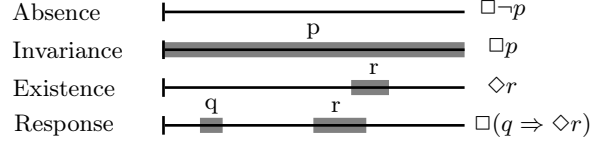


Fig. 4. Sample PLTL patterns for events p , q , r , etc.

trigger. Triggering is an important aspect for this case. Σ , in fact (past) behaviors, is now interpreted as user's preferences to support a new action of a user. The entire input formula for the reasoning process might comprise conjunction of satisfied gt and conjunctions $C(\cdot)$ of the Σ formulas, i.e. cumulatively $gt \wedge C(\Sigma)$. The reasoning process, and its sub-instances, might be performed in a similar way as in the previous case shown in Figure 3.

4 System architecture

The architecture of a proposed system embodied in its well-identified components is briefly discussed in this Section. It allows to understand how the system works, and what are the basic functionalities and services of particular components.

An overall architecture of systems for both model- and learning-based approaches is shown in Figure 5. Signals are gathered (see tracking/sensing in

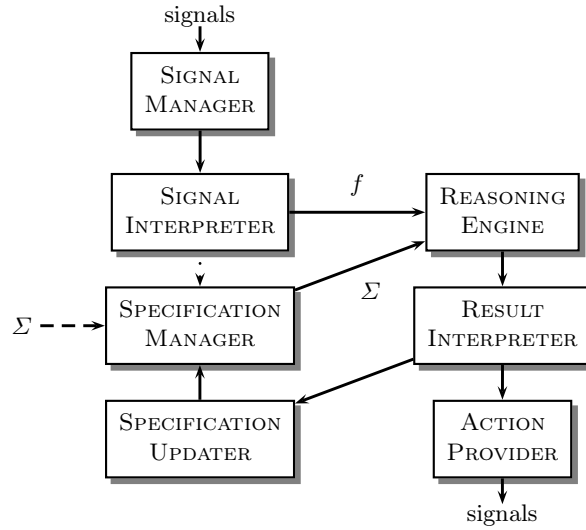


Fig. 5. An overall architecture of systems (flows: solid lines – both approaches, dashed line – only model-based approach, dotted line – only learning-based approach)

Figures 1 and 2) from an environment by Signal Manager. Then signals are in-

interpreted by Signal Interpreter producing temporal logic formulas generated by an algorithm such as Algorithm 1, that is translating events to logical formulas. If once massive amounts of data are processed (the learning-based approach) then formulas flow to Specification Manager that stores the basic logical specification Σ , that is the current logical model of a system. If single data is processed (rather the model-based but also possible in the case of the learning-based approach) then a formula/formulas are provided to the Reasoning Engine. The second input for the Reasoning Engine component is logical specification Σ . The component performs logical reasoning using the semantic tableaux method, however, the resolution-based reasoning is also possible. The output is information, for example, basing on Corollary 1, which is interpreted by Result Interpreter. It provides two outputs. The first one allows to update, if necessary, the current logical specification Σ (stored in Specification Manager) by Specification Updater through deleting or adding some new formulas. The second one allows Action Provider to supply (see reacting/influencing in Figures 1 and 2) signals to the environment. Flows in Figure 5 are not labeled (except specification Σ and formulas/formula f) since they would require precise definitions of the flowing data. On the other hand, their meanings seem intuitive.

Some brief and overall information on methods of Reasoning Engine basing on the semantic tableaux method, see also Section 2 and Corollary 1, is shown in Table 1. $C(\Sigma)$ means a conjunction of all formulas constituting logical spec-

Formulations	Remarks
$C(\Sigma),$ $f \wedge C(\Sigma)$	basic logical properties of a specification, open and closed branches, satisfiability, falsification, contradiction
$C(\Sigma) \Rightarrow f,$ $\neg(C(\Sigma) \Rightarrow f)$	properties that follow, from premises to conclusions, logical consequence, validity, deduction theorem, <i>reductio ad absurdum</i>

Table 1. Methods of Reasoning Engine

ification Σ , in other words, a set of formulas Σ are interpreted (preprocessed) inside Reasoning Engine as a conjunction of formulas $C(\Sigma)$. f is a single formula provided by Signal Interpreter. The reasoning process may comprise many methods and aspects that follow from the input data/formulas, see formulation in Table 1, as well as the assumed reasoning method (truth trees), for example, examining satisfiability of the possessed specification, which happens if a new formula is added to a specification, whether a property can be inferred from a specification using deductive approach, etc.

5 Building and managing specifications

Discovering formal specifications automatically from sensory data streams is discussed below. The process of building logical specifications should be considered

from a broader point of view which follows from the taxonomy discussed at the beginning of Section 3.

The introduction of a method for building logical specifications, the physical world, or smart environment, is formally described over a graph structure.

Definition 1. An attributed graph G is a tuple $\langle V, E, N, \alpha, S, \beta \rangle$, where

- $\langle V, E \rangle$ is a directed graph with a set of vertices V and a set of edges or lines E ,
- N is a set of labels/names,
- $\alpha : V \rightarrow N$ is a function that labels vertices,
- S is a set of labels/sensors, and
- $\beta : V \rightarrow 2^S$ is a function that labels vertices.

A smart environment En is an attributed graph as defined above.

N are commonly used (informal) names for vertices, or nodes (for example: a gate, a crossroad, a staircase, a classroom, etc), if necessary. S are sensors located in a node that detects or measures a physical property and records, indicates, or otherwise responds to it (for example: tactile sensors, temperature, humidity and light sensors, chemical sensors, bio-sensors, etc). This approach enables the gathering of multiple sensory data in a single node, if necessary. For example, on the basis of formal logic, it can be illustrated by a formula

$$s_1 \wedge s_2 \wedge \neg s_3 \wedge s_4 \quad (1)$$

where $s_1, s_2, s_3, s_4 \in S$, and they are responsible for reading sensory data available in a particular node $v_i \in V$, say there are the following four data: temperature exceeded, humidity exceeded, high levels of light, and vibration, respectively. However, to simplify the consideration in the rest of the paper

- the existence of a single sensor in every node is assumed, and
- it is always the object presence sensor/detector that also identifies this object.

Let us consider a set of users/inhabitants $O = \{o_1, o_2, \dots\}$ that operate in a smart environment. These users are identified on-line, i.e. when the system operates, and have unique identifiers. The problem of objects'/users'/inhabitants' unambiguous identification is a well-known question and it may be done in different ways, for example by using RFID, PDA devices, biometric data, image scanning, pattern recognition, and others. The issue of users'/inhabitants' identification is not discussed here.

Events basing on the object presence detection in nodes are registered and the time-stamp for every event is also registered.

Definition 2. An event b_i is a triple that belongs to $\langle O, V, T \rangle$, where

- O is a set of identified users/inhabitants,
- V is a node of a network, and

– T is a set of time stamps.

A behavior B of a smart environment is a set of events $\{b_1, b_2, \dots, b_i, \dots\}$.

For example, $b_i = \langle idEmily, p0018, t2015.02.11.09.30.15 \rangle$ means that the presence of the *idEmily* object is observed at the physical point/area/node *p0018* of the environment, and the time stamp assigned to this event is *t2015.02.11.09.30.15*. Let us note that all nodes that occur in events, or in a behavior, are equivalent to vertices that occur in an attributed graph, or a smart environment. The following notation is introduced. Let $b_i.o_j$ be an object o_j that belongs to an event b_i , and $b_i.v_k$ is a node v_k that belongs to an event b_i , etc.

The algorithm for building logical specifications for every object registered in a smart environment is given as Algorithm 1. *Logical specification* Σ , or L_i ,

Algorithm 1 Building logical specifications for objects O

Input: (New) behavior B (non-empty)

Output: Logical specifications $L_{i=1, \dots}$

```

1: Divide  $B$  into subsets  $B_{i=1, \dots}$  for every object  $o_{i=1, \dots}$ 
2: for every  $B_i$  do
3:    $L_i := \emptyset$  ▷ initiating specification for  $o_i$ 
4:   for  $\forall v \in G$  do
5:     if  $v \notin \{b_i.v_j : b_i \in B_i \wedge j > 0\}$  then
6:        $L_i := L_i \cup \{\Box \neg(G.v)\}$  ▷ saf
7:     end if
8:   end for
9:   Form list  $h = [h_1, \dots, h_n]$  from set  $B_i$ ;
10:  Sort list  $h$  ascending by time stamps;
11:   $l := 1$ ;
12:  repeat
13:     $k := l$ ;
14:    while  $(h_k.v = h_l.v) \wedge (l < n)$  do
15:       $l := l + 1$ ;
16:    end while
17:    if  $h_k.v = h_l.v$  then
18:       $L_i := L_i \cup \{\Diamond(h_k.v)\}$  ▷ liv1
19:    else
20:      if  $h_k.v \neq h_l.v$  then
21:         $L_i := L_i \cup \{\Box(h_k.v \Rightarrow \Diamond(h_l.v))\}$  ▷ liv2
22:      end if
23:    end if
24:  until  $l = n$ 
25: end for

```

is a set of syntactically correct temporal logic formulas. The algorithm bases on

the analysis of all events that occur in a smart environment. The algorithm is explained with the remarks given below.

- Separate specifications for each object are built (line 1);
- Every system should be described using both safety and liveness properties [2];
- It is tested which nodes are not involved in registered events (line 5);
- The most general form for *safety* (informally: nothing bad will ever happen) is $\Box\neg(p)$, i.e. some nodes might be never visited (line 6, labeled “saf”); one can consider the **absence** pattern in terms of Figure 4;
- Auxiliary lists (lines 9 and 10) are created for events that occur for an object;
- List h consists of at least one element (line 9);
- The repeat loop allows to find all sequences of events following each other (lines from 12 to 24);
- The inner loop allows to skip to a different node/event, if any (lines from 14 to 16);
- The most general form for *liveness* (informally: something good will happen) is $\Box(q \Rightarrow \Diamond r)$ or $\Diamond r$, i.e. some nodes are visited (lines 18 or 21, labeled “liv1” or “liv2”, respectively); one can consider the **existence** or **response** patterns in terms of Figure 4, respectively;
- The existence pattern can occur at most once (line 17);
- Summing up, temporal logic formulas are produced in three places of the algorithm which are labeled by “saf”, “liv1”, and “liv2”.

Let us consider the illustrative example for Algorithm 1. Nodes for a smart environment are $\{e2, s03, s07, s08, \}$, i.e. three labeled nodes/vertices. The considered objects/users $O = \{\dots, o_5, \dots\}$. A behavior, that is registered events, is

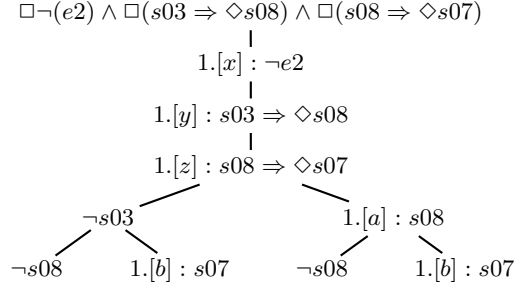
$$\begin{aligned}
 B = \{ & \langle o5, s03, t2015.02.12.09.30.15 \rangle, \\
 & \langle o5, s08, t2015.02.12.09.32.40 \rangle, \\
 & \langle o5, s08, t2015.02.12.09.33.30 \rangle, \\
 & \langle o5, s08, t2015.02.12.09.34.20 \rangle, \\
 & \langle o5, s07, t2015.02.12.09.35.20 \rangle, \\
 & \langle o5, s07, t2015.02.12.11.37.15 \rangle \}
 \end{aligned} \tag{2}$$

The algorithm produces the following logical specification

$$L_i = \{\Box\neg(e2), \Box(s03 \Rightarrow \Diamond s08), \Box(s08 \Rightarrow \Diamond s07)\} \tag{3}$$

Every logical specification can be used for the reasoning process as shown in Figure 3, or in Figure 6 as another example of a truth tree for Formula (3), where conjunction of all sub-formulas are analyzed. Many different methods, as well as deductive systems, for truth trees and semantic tableaux are discussed in work [10] that might help to operate and manipulate efficiently and effectively with truth trees.

The more general remarks for Algorithm 1 are given below.

**Fig. 6.** Another example of a truth tree

- The algorithm produces logical specifications L_i for every object that operates in a smart environment;
- It should be stressed again that, to simplify considerations, the one-sensor case (the object detection) is discussed, in other words, Formula (1) might be replaced by a single atomic sub-formula s_1 as an example, or, in terms of the algorithm, by $h_k.v$ as an example;
- The more general issue is the question when the algorithm should operate, for example, whenever it is required (on demand) or at “the end of a day” (whatever it means), this is an open question for future work;
- Another open issue is the question of what happens when an “old” specification, i.e. specification obtained as a result of the previous execution, is summed, if necessary, with specifications of the current execution, then one should examine the entire specification using decision procedures mentioned at the end of Section 2, as Corollary 1, to discover open and closed branches;
- The sketch for the algorithm that unifies, if necessary, all specifications obtained from Algorithm 1 is given as Algorithm 2, of course, there is no problem to prepare the reverse algorithm, that separates logical specifications due to each object.

Algorithm 2 Building logical specification for smart env. En

Input: Logical specifications $L_{i=1,\dots,n}$ (for object $o_{i=1,\dots,n}$)

Output: Logical specification Σ

- 1: **for** every L_i **do**
 - 2: **for** $\forall f \in L_i$ **do**
 - 3: attribute formula f uniquely due to object o_i
 - 4: **end for**
 - 5: **end for**
 - 6: $\Sigma := \bigcup_{i=1}^n L_i$
-

Corollary 2. *The following two statements are valid.*

1. The time complexity for Algorithm 1 is expressed by $\mathcal{O}(o \cdot n)$, where o is the number of objects that operate in a smart environment, and n is a number of events registered for each object.
2. If a set of all objects and a set of all events are finite, then Algorithm 1 always terminates.

Proof. The main, outer loop depends on a number of objects o . The inner, repeat loop depends on a number of events n . Other operations (assignment) and loops (limited number of iterations) give constant costs. Thus, the time complexity of Algorithm is linearly dependent on the numbers of objects and events.

The number of objects is finite (the for loop), the number of vertices is limited (the inner for loop), as well as the number of registered events is limited (the inner repeat loop), thus, the algorithm always terminates.

Let us supplement this Section with Algorithm 3 that illustrates more formally considerations following Figure 3. It gives an idea how both Reasoning

Algorithm 3 Managing and interpreting truth trees (sketch)

Input: Logical specification Σ ; new formula f

Output: Truth tree Δ ; logical specification Σ ; *Open*;

- 1: $L :=$ formulas Σ that refer to the same object as f refers;
 - 2: Build truth tree Δ for a combined formula $f \wedge C(L)$;
 - 3: $R :=$ select branches of Δ with literals from formula f ;
 - 4: *Open* $:=$ select open branches from R ;
 - 5: *Closed* $:=$ select closed branches from R ;
 - 6:
 - 7: If necessary, remove/modify formulas from specification Σ basing on literals which belong to f and *Closed*;
 - 8: $\Sigma := \Sigma \cup \{f\}$ \triangleright the new basic specification;
 - 9:
 - 10: Analyze nodes from *Open* to provide new actions;
-

Engine and Result Interpretation, shown in Figure 5, work. The $f \wedge C(\Sigma)$ case, see Table 1, is taken into account. It is assumed that initially Σ contains no contradiction. *Closed* is a set of closed branches of a tree and constitutes a base for further modification of the basic logical specification Σ , if necessary, removing formulas that contradict with a newly introduced formula. *Closed'* is a set of all literals extracted from *Closed*. *Open* is a set of open branches of a tree and constitutes a base for selecting satisfiable graph nodes. *Open'* is a set of all literals extracted from *Open*.

If necessary, specification Σ is modified, see lines 7–8, to remove contradictory formulas from a specification. This operation is performed using literals which belong to *Closed/Closed'* (contradictory literals) and f (new formulas, perhaps influencing the basic specification Σ through introducing contradictions, if any),

see the example and the last subcase given below. Analyzing open branches *Open* to provide actions for a system, see line 10, is a standard procedure, see the example and all subcases given below.

The illustrative **example** to supplement both Algorithm 3 and informal considerations succeeding Figure 3 is now provided. For the (Figure) 3.a subcase, $\Sigma = \{\dots, v10 \Rightarrow \Diamond p110, \dots\}$ and $f = v10$. Then $Open' = \{v10, p110\}$ provides literals that allow to find the appropriate formula in Σ , that is formula $v10 \Rightarrow \Diamond p110$. For the 3.b subcase, $\Sigma = \{\dots, ((v11 \Rightarrow \Diamond p115) \vee (v11 \Rightarrow \Diamond p116)), \dots\}$ and $f = v11$. Then $Open' = \{\{v11, p115\}, \{v11, p116\}\}$ provides literals leading to formula $((v11 \Rightarrow \Diamond p115) \vee (v11 \Rightarrow \Diamond p116))$ describes formula showing two equivalent movements (passages $p115$ or $p116$). For the 3.c subcase, $f = \Box(\neg p115)$. Then $Open' = \{v11, p116\}$ and $Closed' = \{\dots, v11, p115, \dots\}$. On one hand, $Open'$ allows to point passage $p116$. On the other hand, $Closed'$, showing literals $v11, p115$, allows to modify a formula as a result of the passage elimination (fire), that is to replace $((v11 \Rightarrow \Diamond p115) \vee (v11 \Rightarrow \Diamond p116))$ by $(v11 \Rightarrow \Diamond p116)$. Then the resulting specification is $\Sigma = \{\Box(\neg p115), \dots, v11, (v11 \Rightarrow \Diamond p116)\}$.

Summing up,

- encoding behaviors to logical specifications is a natural process that can be applied to context-aware systems.
- There are two different approaches mentioned in the beginning of Section 3.
- Some other studies that refer to the implementation and application aspects are open research questions. For example, the form of a formula located in the root of truth trees, that is the disjunction of sub-formulas (the choice between alternatives) or conjunction of sub-formulas (satisfiability, contradiction). Another example is a method for storing formulas, as well as an idea to register multiplicity of formulas/events to introduce additional information about the event popularity.
- Logical specifications, encoding registered behaviors, can be interpreted as preferences understood as a priority in selection. Thus, gathering knowledge about preferences is also expressed as logical formulas.

6 Conclusion

This paper presents a method for behavior discovery as well as the logical satisfiability-oriented reactive analysis for smart and sensor-based environments to support context-aware and pro-active decisions. This approach constructs the process for building logical specifications that fulfill the recognition process providing behavioral specification in terms of temporal logic formulas. The proposed unified logical framework is focused on sensor based activity recognition.

Future works should cover more detailed algorithms, architecture of a multi-agent system and detailed use cases. Considering graph representations and transformations [14,13] is encouraging for efficient implementation and deploying with presented here logical-oriented approach. More comparison study with

other existing methods and more theoretical and experimental evaluations are required for future work.

References

1. Aggarwal, J., Ryoo, M.: Human activity analysis: A review. *ACM Computing Survey* 43(3), 16:1–16:43 (Apr 2011)
2. Alpern, B., Schneider, F.B.: Defining liveness. *Information Processing Letters* 21(4), 181–185 (1985)
3. Chen, S., Liu, J., Wang, H., Augusto, J.C.: A hierarchical human activity recognition framework based on automated reasoning. In: *IEEE International Conference on Systems, Man, and Cybernetics, Manchester, SMC 2013, United Kingdom, October 13–16, 2013*. pp. 3495–3499 (2013)
4. Chen, S., Liu, J., Wang, H., Augusto, J.C.: Formal logical transformation of hierarchical human activity for reasoning based recognition. In: *Decision Making and Soft Computing*, chap. 60, pp. 354–359. World Scientific Publishing (2014)
5. Chomicki, J., Saake, G. (eds.): *Logics for Databases and Information Systems*. Kluwer (1998)
6. Chua, S.L., Marsland, S., Guesgen, H.W.: Behaviour recognition from sensory streams in smart environments. In: Nicholson, A.E., Li, X. (eds.) *Australasian Conference on Artificial Intelligence. Lecture Notes in Computer Science*, vol. 5866, pp. 666–675. Springer (2009)
7. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: *Workshop on The What, Who, Where, When, and How of Context-Awareness (CHI 2000)* (April 2000), <http://www.cc.gatech.edu/fce/contexttoolkit/>
8. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, Los Angeles, CA, USA, May 16–22, 1999. pp. 411–420 (1999)
9. Hähnle, R.: Tableaux and related methods. In: Robinson, J.A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, pp. 100–178. Elsevier and MIT Press (2001)
10. Howson, C.: *Logic with Trees: An Introduction to Symbolic Logic*. Routledge (1997)
11. Klimek, R.: A system for deduction-based formal verification of workflow-oriented software models. *International Journal of Applied Mathematics and Computer Science* 24(4), 941–956 (2014), <http://www.amcs.uz.zgora.pl/?action=paper&paper=802>, web of Science=WOS:000348144200018; IF(2014)=1,227; kwartyle(2014): Q1=applied mathematics, Q3=computer science, artificial intelligence
12. Klimek, R., Kotulski, L.: Proposal of a multiagent-based smart environment for the iot. In: Augusto, J.C., Zhang, T. (eds.) *Workshop Proceedings of the 10th International Conference on Intelligent Environments, Shanghai, China, 30th June–1st of July 2014. Ambient Intelligence and Smart Environments*, vol. 18, pp. 37–44. IOS Press (2014), web of Science=WOS:000360238400006
13. Kotulski, L., Sedziwy, A.: Parallel graph transformations with double pushout grammars. In: Rutkowski, L., et al. (eds.) *Artificial Intelligence and Soft Computing, 10th International Conference, ICAISC 2010, Zakopane, Poland, June 13–17, 2010. Lecture Notes in Computer Science*, vol. 6114, pp. 280–288. Springer (2010)

14. Kotulski, L., Sedziwy, A.: Parallel graph transformations supported by replicated complementary graphs. In: Dobnikar, A., et al. (eds.) *Adaptive and Natural Computing Algorithms - 10th International Conference, ICANNGA 2011*, Ljubljana, Slovenia, April 14-16, 2011, Proceedings. *Lecture Notes in Computer Science*, vol. 6594, pp. 254–264 (2011)
15. Lara, O.D., Labrador, M.A.: A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials* 15(3), 1192–1209 (2013)
16. Magherini, T., Fantechi, A., Nugent, C.D., Vicario, E.: Using temporal logic and model checking in automated recognition of human activities for ambient-assisted living. *IEEE Transactions on Human-Machine Systems* 43(6), 509–521 (2013)
17. Schmidt, R.: Website: accessible theorem provers, <http://www.cs.man.ac.uk/~schmidt/tools/> (2014), accesed on 24-June-2014
18. Wolter, F., Wooldridge, M.: Temporal and dynamic logic. *Journal of Indian Council of Philosophical Research* XXVII(1), 249–276 (2011)